

WRDC-TR-90-8007
Volume VIII
Part 16

AD-A248 924



INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume VIII - User Interface Subsystem
Part 16 - Forms Language Compiler Development Specification

S. Barker

Control Data Corporation
Integration Technology Services
2970 Presidential Drive
Fairborn, OH 45324-6209



September 1990

Final Report for Period 1 April 1987 - 31 December 1990

Approved for Public Release; Distribution is Unlimited

MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

92-10133



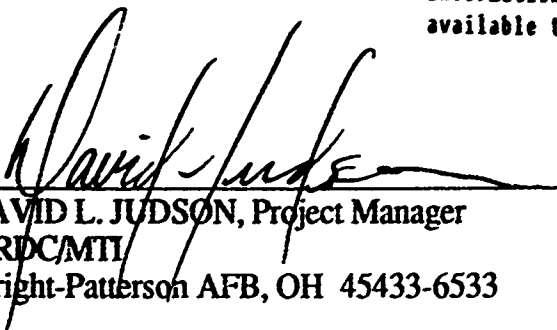
92 4 20 153

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, regardless whether or not the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data. It should not, therefore, be construed or implied by any person, persons, or organization that the Government is licensing or conveying any rights or permission to manufacture, use, or market any patented invention that may in any way be related thereto.

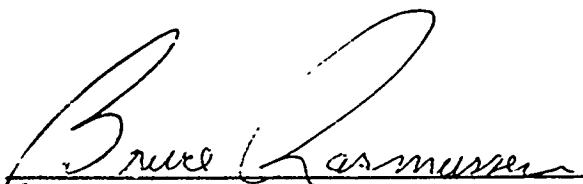
This technical report has been reviewed and is approved for publication.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations


DAVID L. JUDSON, Project Manager
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

FOR THE COMMANDER:


BRUCE A. RASMUSSEN, Chief
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WRDC/MTI, Wright-Patterson Air Force Base, OH 45433-6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS None	
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) DS 620344401			5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-90-8007 Vol. VIII, Part 16	
6a. NAME OF PERFORMING ORGANIZATION Control Data Corporation; Integration Technology Services		6b. OFFICE SYMBOL (if applicable) WRDC/MTI	7a. NAME OF MONITORING ORGANIZATION WRDC/MTI	
6c. ADDRESS (City, State, and ZIP Code) 2970 Presidential Drive Fairborn, OH 45324-6209			7b. ADDRESS (City, State, and ZIP Code) WPAFB, OH 45433-6533	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Wright Research and Development Center, Air Force Systems Command, USAF		8b. OFFICE SYMBOL (if applicable) WRDC/MTI	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM. F33600-87-C-0464	
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB, Ohio 45433-6533			10. SOURCE OF FUNDING NOS.	
11. TIT' See block 19			PROGRAM ELEMENT NO. 78011F	PROJECT NO. 595600
			TASK NO. F95600	WORK UNIT NO. 20950607
12. PERSONAL AUTHOR(S) Structural Dynamics Research Corporation: Barker, S.				
13a. TYPE OF REPORT Final Report		13b. TIME COVERED 4 / 1 / 87 - 12 / 31 / 90	14. DATE OF REPORT (Yr., Mo., Day) 1990 September 30	15. PAGE COUNT 43
16. SUPPLEMENTARY NOTATION WRDC,MTI Project Priority 6203				
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify block no.)	
FIELD	GROUP	SUB GR.		
1308	0905			
19. ABSTRACT (Continue on reverse if necessary and identify block number) This specification establishes the detailed requirements for performance, design test, and qualification of a computer program identified as the Forms Definition Language Compiler. BLOCK 11: INTEGRATED INFORMATION SUPPORT SYSTEM Vol VIII -User Interface Subsystem Part 16 - Forms Language Compiler Development Specification				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson			22b. TELEPHONE NO. (Include Area Code) (513) 255-7371	22c. OFFICE SYMBOL WRDC,MTI

FOREWORD

This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Branch Chief, Integration Technology Division, Manufacturing Technology Directorate, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W. A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. Jimmy P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

SUBCONTRACTOR

ROLE

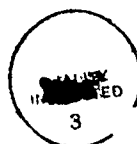
Control Data Corporation	Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS.
D. Appleton Company	Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology.
ONTEK	Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use.
Simpact Corporation	Responsible for Communication development.
Structural Dynamics Research Corporation	Responsible for User Interfaces, Virtual Terminal Interface, and Network Transaction Manager design, development, implementation, and support.
Arizona State University	Responsible for test bed operations and support.

TABLE OF CONTENTS

	<u>Page</u>
SECTION 1.0 SCOPE	1-1
1.1 Identification	1-1
1.2 Functional Summary	1-1
SECTION 2.0 DOCUMENTS	2-1
2.1 Reference Documents	2-1
2.2 Terms and Abbreviations	2-1
SECTION 3.0 REQUIREMENTS	3-1
3.1 Computer Program Definition	3-1
3.1.1 System Capacities	3-1
3.1.2 Interface Requirements	3-2
3.1.2.1 Interface Block Diagram	3-2
3.1.2.2 Detailed Interface Definition	3-4
3.1.2.2.1 Forms Language Compiler Interfaces ...	3-4
3.1.2.2.1.1 Form Definition Language	3-4
3.1.2.2.1.2 Binary Form File Format	3-4
3.1.2.2.1.3 User Input to FLAN	3-5
3.1.2.2.1.4 FLAN Error Messages	3-5
3.1.2.2.2 MAKINC Interfaces	3-6
3.2 Detailed Functional Requirements	3-6
3.2.1 FLAN Functional Requirements	3-6
3.2.1.1 General Requirements	3-6
3.2.1.2 Forms	3-8
3.2.1.3 Graphs	3-9
3.2.1.3.1 Graph Definition	3-10
3.2.1.3.2 Additive Versus Absolute Display	3-11
3.2.1.3.3 Attribute Definitions	3-11
3.2.1.3.4 Axis Information	3-11
3.2.1.3.5 Background Color	3-12
3.2.1.3.6 Color	3-13
3.2.1.3.7 Curve Information	3-13
3.2.1.3.8 Legends	3-13
3.2.1.3.9 Margins	3-14
3.2.1.3.10 Pie Chart Information	3-14
3.2.1.3.11 Text	3-15
3.2.1.4 Graphics Support System	3-15
3.2.1.4.1 2-D Graphics Definition	3-16
3.2.1.4.2 2-D Graphics Attribute Definition	3-16
3.2.1.4.3 2-D Graphics Color Definition	3-17
3.2.1.4.4 2-D Graphics Text	3-17
3.2.1.4.5 2-D Graphics Icons	3-17
3.2.2 MAKINC Functional Requirements	3-18
3.3 Performance Requirements	3-18
3.3.1 Programming Methods	3-18
3.3.2 Program Organization	3-18
3.3.3 Modification Requirements	3-18

SECTION 4.0	QUALITY ASSURANCE PROVISIONS	4-1
4.1	Introduction and Definitions	4-1
4.2	Computer Programming Test and Evaluation ...	4-1
SECTION 5.0	PREPARATION FOR DELIVERY	5-1
APPENDIX A	FORM DEFINITION LANGUAGE SYNTAX	A-1

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



LIST OF ILLUSTRATIONS

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3-1	FLAN Interfaces	3-3
3-2	FLAN Screen	3-5
3-3	Major Internal Data Structures	3-7

SECTION 1

SCOPE

1.1 Identification

This specification establishes the detailed requirements for performance, design, test, and qualification of a computer program identified as the Forms Definition Language Compiler, hereinafter referred to as FLAN. FLAN is one configuration item of the Integrated Information Support System User Interface (IISS UI).

1.2 Functional Summary

FLAN is used to compile Form Definition Language source files into binary Form Definition File format. Form Definition files are used as input to the Form Processor in displaying the forms. MAKINC creates program variable declarations which correspond to the structure of a form and are useful to application programs which make use of Form Processor calls to PDATA and GDATA.

The parser and some procedures of FLAN are used by the Forms Driven Forms Editor (FD FE), the Report Writer (RW), and the Rapid Application Generator (RAP) (all three of which are configuration items). This ensures that the language which is accepted by them is identical.

SECTION 2

DOCUMENTS

2.1 Reference Documents

- [1] Systran, ICAM Documentation Standards, IDS150120000C, 15 September 1983.
- [2] IISS Integration Task Force, Final Report, 1984.
- [3] A. V. Aho and J. D. Ullman, Principles of Compiler Design, Addison-Wesley, 1977.
- [4] S. C. Johnson, "YACC: Yet Another Compiler-Compiler," UNIX Programmer's Manual, Seventh Edition, Vol. 2, Bell Laboratories, 1983.
- [5] General Electric Co., System Design Specification, 7 February 1983.
- [6] Structural Dynamics Research Corporation, Report Writer Development Specification, DS 620244501A, 16 February 1987.
- [7] Structural Dynamics Research Corporation, Rapid Application Generator Development Specification, DS 620244502A, 16 February 1987.
- [8] Structural Dynamics Research Corporation, Text Editor Development Specification, DS 620244600A, 16 February 1987.
- [9] Structural Dynamics Research Corporation, Form Processor Development Specification, DS 620244200A, 16 February 1987.
- [10] Structural Dynamics Research Corporation, Application Interface Development Specification, DS 620244700A, 16 February 1987.
- [11] Structural Dynamics Research Corporation, Forms Driven Form Editor Development Specification, DS 620244402A, 16 February 1987.
- [12] Structural Dynamics Research Corporation, User Interface Services Development Specification, DS 620244100A, 16 February 1987.
- [13] Structural Dynamics Research Corporation, Virtual Terminal Development Specification, DS 620244300A, 16 February 1987.

2.2 Terms and Abbreviations

Application Definition Language: An extension of the Forms Definition Language that includes retrieval of database information and conditional actions. It is used to define interactive application programs.

Attribute: A field characteristic such as blinking, highlighted, black, etc. and various other combinations. Background attributes are defined for forms or windows only. Foreground attributes are defined for items. Attributes may be permanent, i.e., they remain the same unless changed by the application program, or they may be temporary, i.e., they remain in effect until the window is redisplayed.

Common Data Model: (CDM), The IISS subsystem that describes common data application process formats, form definitions, etc. of the IISS and includes conceptual schema, external schemas, internal schemas, and schema transformation operators.

Display List: A list of all the open forms that are currently being processed by the FP or the user.

Field: A two dimensional space on a terminal screen.

Fill Area: A graphic primitive consisting of a single polygon which is filled-in in a specified manner.

Form: A structured view which may be imposed on windows or other forms. A form is composed of fields. These fields may be defined as forms, items, and windows.

Form Definition: (FD), The forms definition language after compilation. It is read at runtime by the Form Processor.

Forms Definition Language: (FDL), The language in which electronic forms are defined.

Form Editor: (FE), The subset of the IISS User Interface that is used to create definitions of forms. The FE consists of the Forms Driven Form Editor and the Forms Language Compiler.

Form Hierarchy: A graphic representation of the way in which forms, items and windows are related to their parent form.

Forms Language Compiler: (FLAN), The subset of the FE that consists of a batch process that accepts a series of forms definition language statements and produces form definition files as output.

Form Processor: (FP), The subset of the IISS User Interface that consists of a set of callable execution time routines available to an application program for form processing.

Graph: A picture correlated with data that alters as the data changes; by necessity, this is a dynamic (not predefined) picture. A graph may be imposed on windows or forms.

Gtext: A graphic primitive consisting of a character string.

Integrated Information Support System: (IISS), A test computing environment used to investigate, demonstrate and test the concepts of information management and information

integration in the context of Aerospace Manufacturing. The IISS addresses the problems of integration of data resident on heterogeneous data bases supported by heterogeneous computers interconnected via a Local Area Network.

Item: A non-decomposable area of a form in which hard-coded descriptive text may be placed and the only defined areas where user data may be input/output.

Message: Descriptive text which may be returned in the standard message line on the terminal screen. They are used to warn of errors or provide other user information.

Operating System: (OS), Software supplied with a computer which allows it to supervise its own operations and manage access to hardware facilities such as memory and peripherals.

Page: An instance of a form that is created whenever a form is added to a window.

Paging and Scrolling: A method which allows a form to contain more data than can be displayed with provisions for viewing any portion of the data buffer.

Polyline: A graphic primitive consisting of a set of connected lines.

Polymarker: A graphic primitive consisting of a set of locations, each indicated by the same type of marker symbol.

Qualified Name: The name of a form, item, or window preceded by the hierarchy path so that it is uniquely identified.

Subform: A form that is used within another form.

User Interface: (UI), The IISS subsystem that controls the user's terminal and interfaces with the rest of the system. The UI consists of two major subsystems: the User Interface Development System (UIDS) and the User Interface Management System (UIMS).

User Interface Development System: (UIDS), The collection of IISS User Interface subsystems that are used by applications programmers as they develop IISS applications. The UIDS includes the Form Editor and the Application Generator.

Window: A dynamic area of a terminal screen on which predefined forms may be placed at run time. Also, a specification of the coordinate system used when defining graphic primitives.

SECTION 3

REQUIREMENTS

3.1 Computer Program Definition

FLAN is a compiler which translates Form Definition Language source files into binary Form Definition File format. The binary Form Definition Files are then used as input by the Form Processor (another configuration item of the IISS UI) for display and entry of data under the control of other application programs.

While FLAN is normally invoked from the IISS Function Screen, another version is available which can be invoked from the host system. This second version is required so configuration management software can be used in managing Forms Definition Language files in a manner similar to other source files.

MAKINC is a program that creates program variable declarations which correspond to the structure of a form and may be used in application programs which make use of the Form Processor calls PDATA and GDATA. The following programming languages are supported: PL/I, COBOL, and C. MAKINC is invoked from the host system.

3.1.1 System Capacities

FLAN is written in the C programming language and is intended to be widely portable across most computer systems.

3.1.2 Interface Requirements

FLAN may be invoked from the IISS function screen or from the host system. In either case the user specifies a Forms Definition Language (FDL) source file to be compiled and FLAN will then output binary Form Definition (FD) files. Error messages are directed to the user's terminal.

The format of the binary Form Definition Files produced by FLAN is constrained to agree with the format expected by the Form Processor configuration item.

The syntax of the Form Definition Language accepted as input is based on the preliminary syntax developed by the IISS Integration Task Force and reported in their Final Report. FLAN also accepts statements of the Report Writer language and the Application Generator language but performs no semantic processing on those statements.

MAKINC is invoked from the host system. The user is prompted for the computer programming language, the name of the output file and for each form for which program variable declarations are desired.

3.1.2.1 Interface Block Diagram

The interface block diagram for FLAN is shown in Figure 3-1. The top box represents the file MYFORMS.FDL which is input to the FLAN compiler (second box). FLAN produces an FD file for each CREATE FORM statement in the source file. Each FD file is input for the Form Processor which is part of the User Interface system. Binary Form Definitions are also an input to MAKINC which produces a file with program variable declarations.

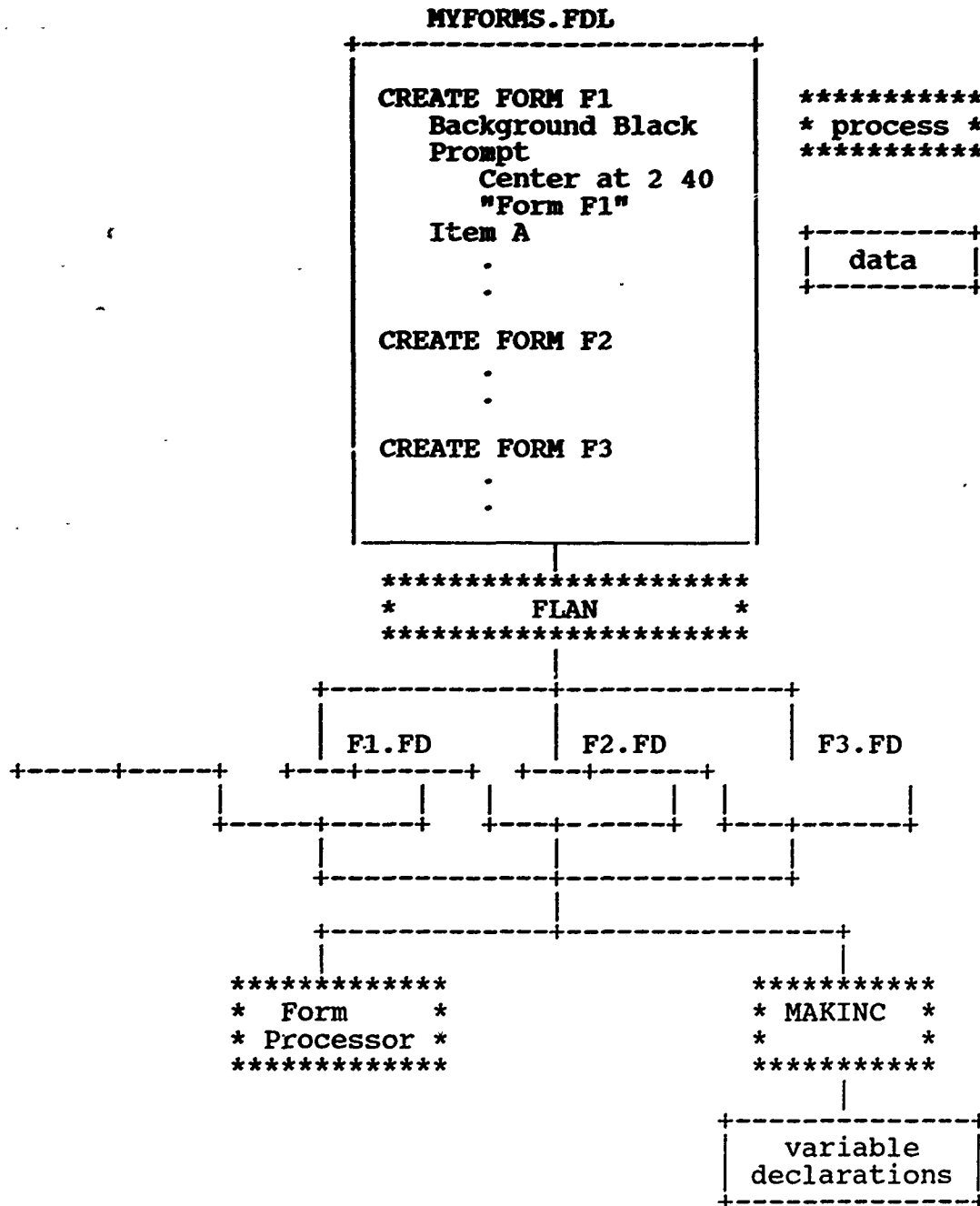


Figure 3-1 FLAN Interfaces

3.1.2.2 Detailed Interface Definition

3.1.2.2.1 Form Language Compiler Interfaces

3.1.2.2.1.1 Form Definition Language

The syntax of the Form Definition Language accepted as input by FLAN is documented in Appendix A. This language is intended to provide access to all Form Processor functionality. It is also intended to be a LALR(1) Grammar for ease of parsing. A number of automatic parser generators are available for LALR(1) grammars, most notably the UNIX utility YACC.

3.1.2.2.1.2 Binary Form File Format

The structure of the records in the binary Form Definition Files produced as output by FLAN are documented in the Form Processor Development Specification. These records are contained in the include file fpd.h. The sequence of records in the file is:

- 1) The version record which identifies the file format.
- 2) A FIELD record.
- 3) The name of the permanent attribute for this field.
- 4) A TEXT record and string, if any. This repeats for each prompt associated with this field.
- 5) An ENODE record that is associated with a predefined value clause, if any. The expression is in a prefix order.
- 6) An ENODE record that is associated with an "appears if" expression, if any. The expression is in a prefix order.
- 7) The help text if this field is an item.
- 8) The ATTMAP record, attribute mapping if this is a form.
- 9) Graphics informational structures, if any. For a graph, more than one structure is present.

3.1.2.2.1.3 User Input to FLAN

The following is the IISS form FLAN used to prompt the user for an input file.

IISS Forms Definition Language Compiler Release 2.0	
Forms Definition Language File Name: _____	
Msg: <u> 0 </u>	application

Figure 3-2 FLAN screen

The host system invocation of FLAN is system dependent. The user specifies a Forms Definition Language (FDL) source file to be compiled and FLAN will then output binary Form Definition (FD) files. Error messages are directed to the user's terminal.

3.1.2.2.1.4 FLAN Error Messages

FLAN error messages are of the format:

line number: type - message text

Where line number is the number of the line on which the error occurred. Type is either WARNING, ERROR, or FATAL. WARNING messages do not prevent FD file generation but indicate potential runtime problems. ERROR messages are sufficiently serious to prevent FD file generation but error checking continues for the rest of the file. FATAL messages prevent all further compilation.

3.1.2.2.2 MAKINC Interfaces

The invocation of MAKINC is system dependent. The user is prompted for the computer programming language, the name of the output file and for each form for which program variable declarations are desired.

3.2 Detailed Functional Requirements

3.2.1 FLAN Functional Requirements

3.2.1.1 General Requirements

FLAN allows one to specify the forms required by the Form Processor. The forms contain the following fields as discussed in detail in the Form Processor Development Specification:

- 1) Fill Area
- 2) Form
- 3) Item
- 4) Polyline
- 5) Polymarker
- 6) Gtext
- 7) Window
- 8) Graph

The forms are specified with a language called the Form Definition Language (FDL). The FDL syntax is described in Appendix A.

FLAN is strictly a transformational process -- its sole function is to translate data from one format to another, both of which are well defined. To accomplish this transformation in the most useful manner (where useful is defined as providing functions which will be of use to other Configuration Items which are currently planned), the following internal data structures are used.

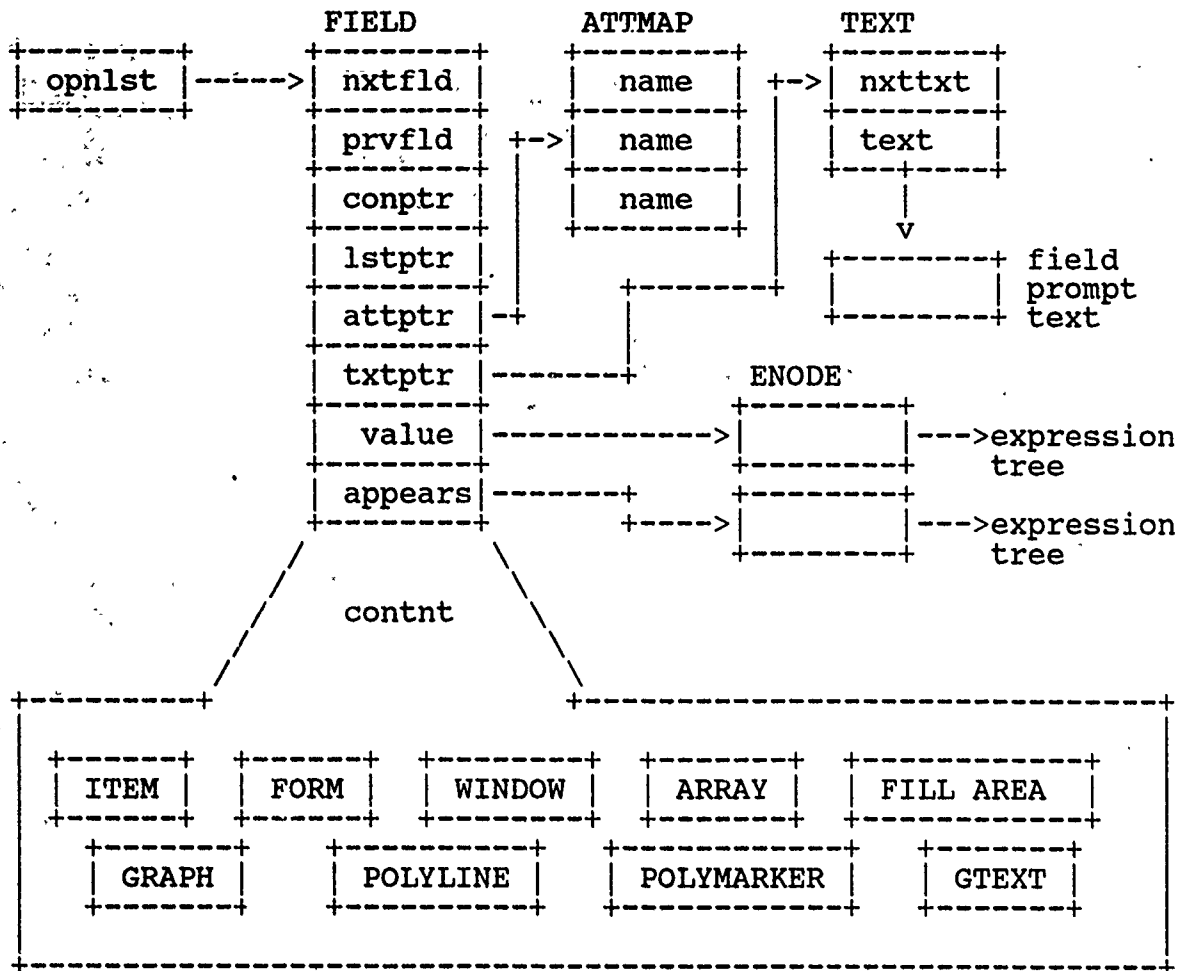


Figure 3-3 Major Internal Data Structures

As statements are read in and recognized, an internal data structure is created and filled in. This data structure is identical to the one used by the Form Processor. Figure 3-3 illustrates this data structure. Note that pointers in the figure which are not shown pointing to anything represent linked lists of items of the type containing the pointer. All of these lists contain zero or more entries depending on the input data.

These linked lists form a binary tree with the `nxtfld` and `prvfld` pointers referring to the next and previous vertices at the same level. The `conptr` and `lstptr` pointers refer to the first and last child nodes of the field. In the case of a child node without siblings, the `conptr` and the `lstptr` of the parent vertex are equivalent. The `contnt` union contains specific information related to the type of field (item, window, etc.). The `ATTMAP` structure consists of the specified form processor attributes for the particular field.

After all of the input statements have been read in and processed and the internal data structure created and filled in without error, the contents of the internal data structure are recorded in binary form files.

When a processing error does occur, an error message is issued to the terminal specifying as much information as possible about the nature of the error and the location of the input statement causing it. When possible, processing continues after an error is recognized so that additional errors may also be detected.

3.2.1.2 Forms

In many applications, there is a large volume of information to be communicated. This can result in many forms and also forms with many items. As the number of forms and item fields per form increases, there may be groups of items that are shared by more than one form or that make sense by themselves. For example, all IRS forms require identification information such as the name, address, and social security number of the taxpayer. These logical groups of items can be made into separate forms and incorporated into other forms as subforms. To incorporate a subform within a form, the area on the host form where the group of items should be located is defined as a form field. A subform within a form is similar to using macros or procedures in a programming language. Forms can be nested to any level that makes sense to keep the form definition for an application as simple and straightforward as possible.

A graph may be viewed as a form. It may be placed within a window or may be incorporated into a form as a subform.

3.2.1.3 Graphs

The Rapid Application Generator (RAP) and the Form Processor (FP) will provide the capability to graphically display data in 2-dimensional format as X-Y plots, bar charts, or pie charts. The FP will provide this through the graphics field, which will contain a single graph of pre-defined type and structure. However, the associated graphics picture will be dynamic. In the case of X-Y plots and bar charts, more than one dataset may be plotted in a single graph to yield multiple curves or bars. An X-Y plot or bar chart may contain both additive and absolute dependent datasets for any given independent dataset. If both additive and absolute dependent data are present, the additive sets will be associated with the specified dependent dataset. The default dependent dataset to be added to is the first defined.

Size and location values are stated in terms of the default terminal character sizes and positions. No scaling is done automatically; the user has the responsibility of ensuring that information is of the proper size to fit in the available space. Portions of a picture which do not fit in a displayed space are clipped, not wrapped. The FP will attempt to adjust pictures appropriately for the aspect ratio of the device on which they are displayed.

Optional attributes will have default values set so that a graph can be included in a display with a minimum of effort at specification. Attributes unsupported on a given device (ex. color on a monochrome monitor) will default to appropriate values.

On devices supporting them, capabilities shall include a choice of line style and width (solid, dashed, dotted, dashed-dotted, and varying factors of thickness), and the choice of color for the curve, the background, and the area under the curve. Other available choices shall include the specification of the size and font style for text in axis labels, tick mark labels, legend entries, the lower and upper limits of axes, the scale (linear, logarithmic) of each axis, the presence or absence of a grid, the locations and lengths of the axes, some shading patterns below the curves, and symbols to appear at the data points. For graphs consisting of more than one curve there shall also be the option of displaying the dependent data additively or absolutely as measured on the axis.

Graphs shall be output display only, with the exception that data points may be picked on the graph. The graphics picture will then be altered to retain only the picked area.

The data sources, if specified, for the graphs denoted by independent and dependent data in the language syntax must be either numeric item fields or constants. The item fields may or may not be displayed. If an item field is an input field, data in it can be altered so that the next time the graph which uses that field is displayed it will reflect the new value. It should be noted that if more than one value is associated with either an independent or a dependent dataset, and the source is an item field, the source may be defined as either a repeating numeric item field, or individual item fields. The repeating set of fields is preferred.

If an item will be receiving data from an NDML SELECT statement, then the form containing the item must be a repeating form field. The path designation must contain the form name with a "*" qualifier in order to be consistent with the RAP.

If no item fields are specified from which to obtain the data, the data must be listed in the USING and CURVE clauses of the graph definition. If more than one independent dataset is to be used, the independent dataset must be specified in the VERSUS clause of the CURVE definition. The USING clause indicated that the listed independent dataset is to be used for a curve that does not contain a VERSUS clause.

3.2.1.3.1 Graph Definition

The graph is defined as a graphics field on a form in the same way as other fields are defined. The graph field must also appear in a CREATE GRAPH statement. The size and location are required parts of the graph definition. The location of the graph can be defined relative to other fields in the form containing the graph. Prompts associated with the graph are external to the graph as they are for item fields.

3.2.1.3.2 Additive Versus Absolute Display

When data are displayed using X-Y plots, more than one dependent dataset may be plotted in a given graph. Two or more curves are distinguished by differing color and/or linestyle. Dependent values can be measured on the dependent axis, either relative to the independent axis or relative to a curve already displayed. The former method is absolute display, the latter is additive display. By default, displays will be absolute.

3.2.1.3.3 Attribute Definitions

Attribute bundles may be defined and named so that they can be easily referenced elsewhere in the graph definition. For text, attribute bundles may include the primitives FONT, COLOR, SIZE, and UPVECTOR. UPVECTOR specifies the up direction of the individual characters. The characters are placed along the line perpendicular to the character UPVECTOR. For curves, attribute bundles may include the primitives LINE_TYPE, LINE_WIDTH, COLOR, SYMBOL, and SYMBOL_FREQUENCY. SYMBOL_FREQUENCY instructs the FP to place a symbol at the first, last, and every nth datum position. The default frequency is one. If both LINE_TYPE and SYMBOL are specified, the data curve will be drawn and markers will then be placed over the curve at the indicated frequency. For curves, SIZE refers to the height of the marker.

3.2.1.3.4 Axis Information

Axis and tick mark labels can be specified separately for the dependent and independent axes. The text in these labels can be defined to have any available font, color, and size (in units of the terminal's standard character height and width). By default, text will have a size of 1 by 1 and a simple block font; the color will be the default contrasting color to the background (ex. white on black). The color of the axis itself including tick marks can also be specified and may differ from the color of the labels.

For a bar or an X-Y plot, if an axis is not specified, a default axis will be created based on the maximum and minimum data values associated with that axis.

The tick mark clause specifies the step between or the number of major tick marks and optionally the number of minor divisions between each pair of major tick marks. The tick mark label is comprised of a sequence of strings. If the number of strings is less than the number of major tick marks, then the tick mark labels are reused from the beginning. If the tick mark label is not specified and the maximum and minimum axis values are numeric, the labels will be generated.

If no label string is defined for an axis, the associated item field name is used as the label. In a case where more than one item field is associated with a vertical axis, the name of the first such item field is used. If no tick marks are defined, then the axis is divided into a number of intervals which yield nice tick mark label values. Nice values are defined to be multiples by powers of ten of values in the set (.1, .2, .25, and .5).

The lower and upper limits and the scale (linear or logarithmic) of the axes may also be specified independently. If the range of the data exceeds the limits specified, then data outside of the range are not displayed. Unspecified limits are set by default to give the smallest range which will span the entire range of the data such that the tick mark labels can be either those given by the user or those in the set of nice values as defined in the paragraph above. If the scale is not defined to be logarithmic, it is linear by default.

X-Y plots and bar charts can be defined to have grids superimposed upon them. The default choice is no grid displayed. The grid is comprised of lines of constant value of the coordinate specified. If minor tick marks have been specified, then a fine grid will connect to minor tick marks. If minor tick marks have been defined for only one axis, then fine grid lines will appear perpendicular only to that axis.

3.2.1.3.5 Background Color

The background of the graph field can be defined to have any available color.

3.2.1.3.6 Color

On terminals supporting color, the following colors are defined: red, yellow, green, cyan, magenta, blue, white, and black. Unless the defaults are used, it is the user's responsibility to ensure visibility of the graph components by selecting contrasting colors. Improper choice of colors may render some components invisible. For example, defining the background as white and using white to draw the axes and tick marks will make the axes indistinguishable from the background.

3.2.1.3.7 Curve Information

For X-Y plots, several attributes of the curves may be specified, including the color of the curve as well as the color or shading pattern of the area below the curve. On devices supporting the feature, the line type (solid, dashed, etc.) and line width may also be specified. It is possible to specify an additional optional pattern that is to be used if the physical device is monochromatic using the OR clause of the SHAPE or DISPLAY AS specification. For X-Y plots, symbols may optionally be placed at data point locations using a specified frequency, either in addition to or instead of the curve. By default, data points are connected by solid line segments in the order of occurrence of the source item field values, and no data point symbols appear. The symbols may be chosen from a catalog of pre-defined symbols and will include minimally the dot, plus sign, asterisk, cross or x, and circle. By default, no shading occurs below curves and the color is the background color.

3.2.1.3.8 Legends

An X-Y plot, bar chart, or pie chart can be defined to have a legend in a specified location. By default, no legend will appear. The legend can optionally be circumscribed by a box. The location value is the location of the upper left-hand corner of the legend, relative to the graph. The legend entry is defined in the CURVE clause or the PIE clause of the CREATE GRAPH statement. The attribute name associated with the legend may be used to specify the font, size, and color attributes for the legend entry text.

3.2.1.3.9 Margins

The amount of space surrounding the graph which is delimited by the axes is adjustable. This space can be set by the location specifications for the axes in the CREATE GRAPH statement. By default, all margins are 10 percent.

3.2.1.3.10 Pie Chart Information

The data source for a pie chart, if given, may be a single repeating item field or list of item fields. If the repeat factor is one or the list consists of one item field, then the pie chart consists of one circular segment. The segments correspond in order to the elements of the item field. If the data source is not indicated, a constant list may be given.

Some pie chart attributes are associated with individual segments. The segments are numbered sequentially in counter-clockwise fashion from one beginning at a horizontal radius vector to the right.

The explosion factor, which is applied on a segment by segment basis, is the percentage of the radius of the pie by which a segment is projected radially outward. By default, no segments are exploded.

Individual segments may be labeled with a string of specified font, size, and color. The labels may be placed wither inside or outside of the segment. Alternatively, legend entries may be defined for each segment.

The percentages of the whole may be displayed for each segment, either inside or outside of the segment.

The item field values may be displayed for each segment, either inside or outside of the segment, by requesting the quantity.

Each segment may be individually colored or shaded with a pattern. By default, no pattern will appear with the segments and the interior color will be the background color of the gtraph field. It is possible to specify an additional optional pattern that is to be used if the physical device is monochromatic using the OR clause of the SHADE specification.

3.2.1.3.11 Text

Text may be included anywhere within the graph field for titles, annotation, etc., via the LABEL clause. The GDL LABEL clause is similar to the FDL PROMPT clause with the exception that the text attributes include FONT, SIZE, and, if supported by the device, COLOR and UPVECTOR. UPVECTOR specifies the up direction of the individual characters. The characters are placed along the line perpendicular to the character UPVECTOR.

3.2.1.4 Graphics Support System

The Graphics Support System is a subsystem of the User Interface which may be accessed either directly or indirectly by an application program. The system is to support both static and dynamic graphics in two or three dimensions.

The static graphic support allows for the display of predefined 2-D images (icons) in conjunction with text (i.e. as part of a form). The 2-D graphics primitives contained in a form are analogous to items. These images may be specified in the application FDL source file. A form containing 2-D graphics may be picked (as any form can) allowing an ICON to be picked.

The dynamic graphics support (PHIGS) allows for the display of variable images, the general form of which may be predefined. A structure can contain a number of primitives including references to other structures just as a form can contain items and prompts as well as references to other forms. Dynamic graphics support is provided only through services in the Form Processor.

The geometric primitives include polyline, polymarker, gtext, and fill areas. Primitives also exist to set geometric primitive attributes such as color, line type, line width, font, character spacing, text alignment, interior style, and interior color. Additional subroutines for the dynamic support, set modeling and viewing parameters, retrieve the current values of various internal variables, and control the overall operation of the Graphics Support System.

3.2.1.4.1 2-D Graphics Definition

Two dimensional graphics images can be composed of Polyline, Polymarkers, Gtext, and Fill Areas. The graphics primitives are analogous to the items contained in a Form. Thus a Form contains the primitives. The primitives are defined in the form with the primitive name (ie, POLYLINE) followed by a unique identifier. Each of the primitives must have a location and size specified. The primitives have the following capabilities:

Polyline: A polyline is a primitive consisting of a set of connected lines. The polyline is defined as points which specify the start and end points of each line. Example, a polyline may have two lines, the starting location for the first line, the end location for the first (and start location for the second), and the end location for the second. Thus a total of three locations will define a polyline of two lines.

Polymarker: A polymarker is a graphic primitive consisting of a set of locations, each location indicated by the same type of marker symbol. The polymarker is defined as points which specify the location for the markers. The points can be repeated.

Text: Graphics text is an implementation dependent text which can have attributes which are defined in a manner other than the normal forms character or terminal character set. Text can be defined to have a greater flexibility in selection of font, spacing, color, the path and up direction, as well as the alignment and location.

Fill Area: A fill area is an implementation dependent graphics primitive which consist of a single polygon which is filled-in with a specified color or pattern. A fill area is defined by specifying the location of points which will bound or enclose the area. A style of "fill" is also specified.

3.2.1.4.2 2-D Graphics Attribute Definition

The attributes of the 2-D graphics primitives are specified as STYLE, SIZE, and COLOR.

Polyline: The STYLE, is specified as SOLID, DASHED, DOTTED, or DASHED DOTTED. The SIZE (line width) is specified as a real number.

Polymarker: The STYLE, is specified as DOT, PLUS, STAR, CIRCLE, or CROSS. The SIZE (approximate diameter) is specified as a real number.

GTEXT: Is specified as the following:

- o Font - style expressed as an integer
- o Precision - may be STRING, CHARACTER, or STROKE
- o Expansion - expansion of characters expressed as a real
- o Spacing - spacing between characters expressed as a real

FILL AREA: The **STYLE** is specified for a region which may be of **HOLLOW**, **SOLID**, a **PATTERN**, a **HATCH**, or **EMPTY**. The **PATTERN** and **HATCH** styles must be followed by an integer to specified the appropriate style. The **SIZE** specification is not used with the fill area.

3.2.1.4.3 2-D Graphics Color Definition

On terminals supporting color, the following colors are defined: red, yellow, green cyan, magenta, blue, white and black. Unless the defaults are used, it is the user's responsibility to ensure visibility of the graphics primitives by selecting contrasting colors. Improper choice of colors may render some components invisible. For example defining the background as white and using white to draw polylines, fill areas, etc. would make the primitives indistinguishable from the background.

3.2.1.4.4 2-D Graphics Text

In addition to allowing the **STYLE** (see 3.2.1.4.2 2-D Graphics Attributes), Text can have additional attributes to specify the **SIZE** of the characters, the **UP** vector (specifies the up direction of the individual characters), **PATH** to follow (ie, allow text to be viewed along a oblique angle with respect to some part). The **ALIGNMENT** of the character within the character cell itself can be specified as (horizontal) left, center, right, and (vertical) as top, cap, half, base, bottom.

3.2.1.4.5 2-D Graphics Icons

An Icon is a form which is an image composed of 2-D graphics primitives (and graphics Text). An **ICON** (as any form can) may be picked to allow the representation of an action.

3.2.2 MAKINC Functional Requirements

MAKINC is strictly a transformational process -- its sole function is to translate data from one format to another, both of which are well defined. The input to MAKINC is constrained to agree with current version binary Form Definition files accepted by the Form Processor. The output is a set of program variable declarations which are syntactically and semantically correct for the language of choice and which correspond (in structure, name and size) to the forms and fields in the FD file. These programming languages include:

- 1) C
- 2) COBOL
- 3) PL/I
- 4) FORTRAN

3.3 Performance Requirements

3.3.1 Programming Methods

A parser generator, YACC, is used to create the parser and existing Form Processor routines are used as appropriate as the internal data structures used by FLAN are identical to that used by the FP.

3.3.2 Program Organization

The actual module structure was refined as other CIs were developed which make use of the functionality of FLAN. FLAN basically consists of a lexical analyzer, an LALR(1) parser, some semantic procedures and a code generator (writes binary Form Definition files). Syntax errors are detected by the lexical analyzer and parser. Semantic errors are detected by the semantic procedures.

3.3.3 Modification Requirements

The use of a parser generator makes changes to the Form Definition Language easy to implement since one specifies only the grammar and not the parse tables. Using existing Form Processor routines ensures that changes made to FP data structures and procedures will require few changes be made to FLAN's procedures.

SECTION 4

QUALITY ASSURANCE PROVISIONS

4.1 Introduction and Definitions

"Testing" is a systematic process that may be preplanned and explicitly stated. Test techniques and procedures may be defined in advance and a sequence of test steps may be specified.

"Debugging" is the process of isolation and correction of the cause of an error.

"Antibugging" is defined as the philosophy of writing programs in such a way as to make bugs less likely to occur and when they do occur, to make them more noticeable to the programmer and the user. In other words, as much error checking as is practical and possible in each routine should be performed.

4.2 Computer Programming Test and Evaluation

The quality assurance provisions for test consists of the normal testing techniques that are accomplished during the construction process. They consist of design and code walk-throughs, unit testing, and integration testing. These tests are performed by the design team. Structured design, design walk-through and the incorporation of "antibugging" facilitate this testing by exposing and addressing problem areas before they become coded "bugs".

The integration testing entails the use of a test application. This test program displays forms, reads input from forms, and displays results.

Each function is tested separately, then the entire subsystem is tested as a unit. All testing is done on the IISS test bed.

SECTION 5

PREPARATION FOR DELIVERY

The implementation site for the constructed software is the ICAM Integrated Support System (IISS) Test Bed site located at Arizona State University, Tempe, Arizona. The software associated with each CPCI release is delivered on a media which is compatible with the IISS Test Bed. The release is clearly identified and includes instructions on procedures to be followed for installation of the release. Integration with the other IISS CPCIs will be done on the IISS TEST BED on a scheduled basis.

APPENDIX A

FORM DEFINITION LANGUAGE SYNTAX

This document uses the following notation to describe the syntax of the FDL entries:

UPPER-CASE	Identifies reserved words that have specific meanings in the FDL. These words are generally required unless the portion of the statement containing them is itself optional.
lower-case	Identifies names, numbers, or character strings that the user must supply.
Initial upper-case	Identifies a statement or clause that is defined later on.
_ Underscores	Identify reserved words or portions of reserved words that are optional.
() Braces	Indicate that one of the enclosed vertically stacked options is required.
[] Brackets	Indicate that the enclosed clause or option is optional. When two or more options are vertically stacked within the brackets, one or none of them may be specified.
... Ellipsis	Indicates that the preceding statement or clause may be repeated any number of times.
field_name	means the qualified name that identifies a form, window, or item field. The specific field type replaces the word "field" when appropriate.

Form_Definition

```
CREATE FORM form_id
    [ ATTRIBUTE attribute_id ( Primitive [ , ... ] ) ] ...
    [ SIZE int [ BY int ] ]
    [ BACKGROUND attribute_id ]
    [ PROMPT Location string ] ...
    [ Field_Definition ] ...
```

Field_Definition - Fillarea

```
FILL AREA fill_id
    [
        STYLE
        {
            HOLLOW
            SOLID
            PATTERN int
            HATCH int
            EMPTY
        }
    ]
    COLOR color
    POINTS G_location ...
    [APPEARS IF Expression]
```

Field_Definition - Form

```
FORM form_id [ ( Repeat_Spec [ , ... ] ) ]
    Location
    SIZE int [ BY int ]
    [ BACKGROUND attribute_id ]
    [ PROMPT Location string ] ...
    [ APPEARS IF Expression ]
```

Field_Definition - Graph

```
GRAPH graph_id
    Location
    SIZE cols [ BY rows ]
    [ PROMPT Location string ] ...
```

[APPEARS IF Expression]

Field_Definition - Item

ITEM item_id [(Repeat_Spec [, ...])]

Location

[SIZE int [BY int]]

[VALUE Expression]

DISPLAY AS attribute_id

[[LEFT]

[UPPER]

| DOMAIN (| RIGHT |

[MUST ENTER]

[[MUST FILL]

[LOWER]

[NUMERIC] [MAXIMUM int] [MINIMUM int]) }

[(string)]

[HELP { name }]

[(APPLICATION)]

[PROMPT Location string] ...

[APPEARS IF Expression]

Field_Definition - Polyline

POLYLINE polyline_id

```
[
  STYLE {
    int
    SOLID
    DASHED
    DOTTED
    ( DASHED DOTTED )
  }
]
```

[SIZE real]

COLOR color

POINTS G_location ...

[APPEARS IF Expression]

Field_Definition - Polymarker

POLYMARKER polymark_id

```
[
  (
    int
    DOT
    PLUS
    STYLE {
      STAR
      CIRCLE
      CROSS
    }
  )
]
```

[SIZE real]

COLOR color

POINTS G_location

[APPEARS IF Expression]

Field_Definition - Gtext

GTEXT text_id

[FONT int]

[PRECISION { STRING
CHARACTER }]
[STROKE]]

[EXPANSION real]

[SPACING real]

COLOR color

[SIZE real]

[UP real real]

[()]
RIGHT
|
LEFT
| PATH { } |
UP
|
DOWN
[()]

[ALIGNMENT { LEFT
CENTER
RIGHT } [TOP
CAP
HALF
BASE
BOTTOM]]

G_location string

[APPEARS IF Expression]

Field_Definition - Window

```
WINDOW window_id [ ( Repeat_Spec [ , ... ] ) ]  
    Location  
    SIZE int [ BY int ]  
    [ BACKGROUND attribute_id ]  
    [ PROMPT Location string ] ...  
    [ APPEARS IF Expression ]
```

Graph Definition

```
CREATE ( BAR )  
      ( PIE ) GRAPH graph_id  
      ( LINE )  
    [ Location ]  
    [ SIZE int BY int ]  
    [ USING ( [ independent_data ] [ AXIS axis_id ] ) ]  
    [ ATTRIBUTE attr_id gtype ( Graph_Prim [ , ... ] ) ] ...  
    [ LEGEND [ HORIZONTAL ] Location [ BOX ] ]  
    [ [ VERTICAL ] ]  
  
    [ LABEL [ Location, [DISPLAY AS attr_id] ] ] string ...  
    [ [ DISPLAY AS attr_id, Location ] ]  
    [ BACKGROUND attribute_id ]  
    [ Graph_Field_Definition ] ...
```

Graph_Field_Definition - Axis

AXIS axis_id

DISPLAY AS attribute_id

(HORIZONTAL)

(VERTICAL)

()

[LABEL label_id [Location] string] ...

[Location]

[MINIMUM lower_limit]

[MAXIMUM upper_limit]

[()]

| SCALE (LINEAR) |

| LOG) |

[()]

[SIZE length]

[TICK [EVERY] ndiv [minor] tick_id string ...]

[[FINE] GRID]

Graph_Field_Definition - Bar or Curve

CURVE curve_id

dependent data [USING AXIS axis_id]

[VERSUS independent data]

[DISPLAY AS attribute_id [OR attribute_id]]

[LEGEND attribute_id string]

[SHADE [COLOR color] [PATTERN pattern]

[OR pattern]]

[ABSOLUTE]

[ADDITIVE [USING CURVE curve_id]]

Graph_Field_Definition - Pie Segment

PIE int

```
[ EXPLODE real ]  
[ LABEL attribute_id [ Location ] string ]  
[ LEGEND attribute_id string ]  
[ PERCENT { ( INSIDE )  
             { OUTSIDE } attribute_id  
             ( Location ) }  
[ QUANTITY { ( INSIDE )  
              { OUTSIDE } attribute_id  
              ( Location ) }  
[ SHADE [ COLOR color ] [ PATTERN pattern ]  
                                                [ OR pattern ] ]
```

Primitive

```
( BACKGROUND Color )  
  DISPLAY Color  
  BOLD  
  DIM  
  UNDERSCORE  
  SLOWBLINK  
  FASTBLINK  
  HIDDEN  
  GUARDED  
  REVERSE  
  ORED  
  TABFIELD  
( NOWRITE )
```

gtype

```
( LINE )
( FILL )
( PROMPT )
( MARKER )
```

Graph_Prim

```
( DISPLAY color )
FONT int
STYLE ( int )
      ( SOLID )
      ( DASHED )
      ( DOTTED )
SIZE real
      ( int )
      ( DOT )
      ( PLUS )
SYMBOL ( STAR ) [ [ SYMBOL_FREQUENCY ] frequency ]
      ( CIRCLE )
      ( CROSS )
      ( )
( UPVECTOR angle )
```


Color

```
(
  BLACK
|
| RED
|
| GREEN
|
| YELLOW
(
  BLUE
|
  MAGENTA
|
  CYAN
|
  WHITE
(
  )
```

Location

```
[ Rpt ] AT ( [ int ] [ int ] [ RELATIVE TO [ Rpt OF ] ] )
            { [ field_name ] }
            ( Partial_loc [ AND Partial_loc ] )
```

Partial_loc

```
( ROW int )
| COLUMN int |
| ( LEFT OF ) |
| ( RIGHT OF ) |
| [ int ] { [ Rpt OF ] [ field_name ] } |
| ABOVE |
| BELOW |
( ( ) )
```

G_location

```
AT ( [ real ] [ real ] [ RELATIVE TO [ Rpt OF ] ] )
    { [ field_name ] }
    ( Partial_g_loc [ AND Partial_g_loc ] )
```

Partial_g_loc

```
( ROW real )
| COLUMN real |
| ( LEFT OF ) |
| ( RIGHT OF ) |
| [ real ] ( ABOVE ) [ Rpt OF ] [ field_name ] |
| ( BELOW ) |
( ( ) )
```

Rpt

```
( TOP LEFT )
{ TOP
  TOP RIGHT
  LEFT
  CENTER
  RIGHT
  BOTTOM LEFT
  BOTTOM
  BOTTOM RIGHT }
```

Repeat_Spec

```
( int-1          )
| int-1 / int-1 | ( HORIZONTAL )
{ int-1 / int-2 } { VERTICAL } [ WITH int SPACES ]
| int-1 / *      | (          )
( *              )
```

Expression

```
( string
  field_name
  int
  ( Expression )
  - Expression
  NOT Expression
  Expression Binop Expression
  Expression ? Expression : Expression
  { INDEX (field_name)
    GETATT (field_name, type)
    GPAGE (field_name, page)
    GWINDO (field_name)
    APPEARS (field_name)
    BETWEEN (Expression, Expression, Expression)
    CURSOR (field_name)
    IN (Expression, Expression, ...)
  }
  ( ROLE (Expression) )
```

Binop

```
( | | )
  +
  -
  *
  /
  =
  { !=
    <
    <=
    >
    >=
    AND
  }
  ( OR )
```

System Defined Attribute Names

```
ERROR (ored, fastblink)
GUARDED (ored, guarded)
TEXT (guarded)
```

INPUT (reverse, tabfield)
OUTPUT (bold, guarded)
HIDDEN (hidden, reverse, tabfield)
CALCULATED (bold, guarded, nowrite)
TABFLD (guarded, tabfield)
XPARNT ()
BLACK (background black, display white)
RED (background red, display white)
GREEN (background green, display black)
YELLOW (background yellow, display black)
BLUE (background blue, display white)
MAGENTA (background magenta, display white)
CYAN (background cyan, display black)
WHITE (background white, display black)